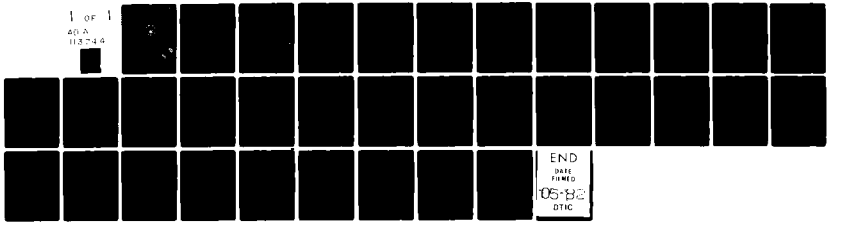
AD-A113 744 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 9/2
NAVLISP REFERENCE MANUAL.(U)
MAR 82 A D SAMPLES
UNCLASSIFIED NPS52-82-004 NL

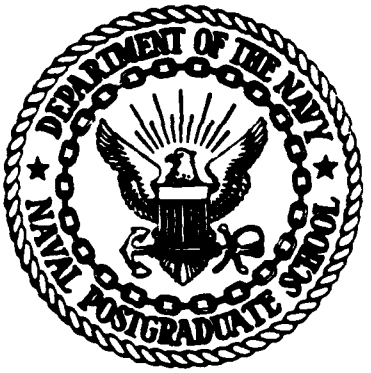


AD A113744

NPS52-82-004

NAVAL POSTGRADUATE SCHOOL

Monterey, California



Navlisp Reference Manual

A. Dain Samples

March 1982

DTIC ELECTE APR 23 1982 S D H

Approved for public release; distribution unlimited

Prepared for:

Chief of Naval Research
Arlington, Va 22217

DTIC FILE COPY

82 04 23 025

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral J. J. Ekelund
Superintendent

David A. Schrady
Acting Provost

The work reported herein was supported in part by the Foundation Research Program of the Naval Postgraduate School with funds provided by the Chief of Naval Research.

Reproduction of all or part of this report is authorized.

This report was prepared by:

A. DAIN SAMPLES
Computer Programmer
of Computer Science

Reviewed by:

GORDON H. BRADLEY, Chairman
Department of Computer Science

Released by:

WILLIAM M. TOLLES
Dean of Research

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS52-82-004	2. GOVT ACCESSION NO. AD-A113 744	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Navlisp Reference Manual		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) A. Dain Samples		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61152N; RR000-01-10 N0001482WR20043
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		12. REPORT DATE March 1982
		13. NUMBER OF PAGES 35
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Chief of Naval Research Arlington, Va 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

LISP

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Naval Postgraduate School's Computer Laboratory has developed a dialect of LISP, called Navlisp, to run under PWB/UNIX on a PDP11/50. This manual is not a tutorial to LISP. It is intended for those who know the basics of the LISP programming language and wish to use the Navlisp dialect.

DTIC
APR 23 1982
S H

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Navlisp X402
Reference Manual V1.2
A. Dain Samples
March 17, 1982

Introduction

The Naval Post-Graduate School's Computer Laboratory has developed a dialect of LISP, called Navlisp, to run under PWB/UNIX on a PDP 11/50. While loosely based on Maclisp, Navlisp is unlike other LISP systems in several important ways.

Like all processes that run under UNIX, Navlisp interfaces easily to other processes. Since many functions normally found in LISP systems (e.g. LISP oriented editors) are available through the UNIX environment, Navlisp is much smaller than conventional LISP systems (approximately 36K of code) and yet has full LISP functionality.

Navlisp is also different from other LISPs in the design of the language. A new control structure, Parnas' it-ti*, has been implemented as an enhancement to the standard LISP cond. Surprisingly, this modification is compatible with the old definition of cond, but is much more powerful: with the let function, the new cond is powerful enough that the very unstructured (and anachronistic) prog feature of standard LISP is not necessary, and has not been implemented in Navlisp.

In addition, a list element accessing function suggested by Dana Scott has been implemented: (n list) will access the nth element of list, where n is an integer (see the description of the 0 function).

This manual is not a tutorial to LISP. It is intended for those who know the basics of the LISP programming language and wish to use the Navlisp dialect. The descriptions contained herein are brief and assume at least a passing knowledge of both LISP and UNIX. Winston and van Horn's LISP book provides a good tutorial to the language, and is highly recommended.

1. Running Navlisp

Navlisp maintains a user "workspace" in which user defined objects are maintained in internal format. This workspace is loaded each time Navlisp is executed, and saved each time the user exits Navlisp. The workspace must be initialized in your directory before running Navlisp. To do this, run initnavlisp

* For a more detailed explanation of the it-ti, see "Implementation of Parnas' it-ti Construct in LISP", A. D. Samples, Computer Science Laboratory Technical Report NPS52-82-005, Naval Postgraduate School, 1982.

DTIC COPY INSPECTED 2

Accession For
NTIS GRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A

first:

```
% initnavlisp     : do only once

% navlisp         : now running
```

Currently, the default workspace file name is .navlisp in the current directory. In anticipation of future enhancements, the user is able to specify a different workspace file name on the command line for initnavlisp or navlisp. This file will then be the target of SAVE, RESTORE, SCRUNCH, and bye (q.v.). Currently, the file name cannot be changed while navlisp is running, although that is a planned enhancement. E.g., to use ../mywork as the workspace storage file:

```
% initnavlisp ../mywork

% navlisp ../mywork
```

Some development work will continue on Navlisp. In the future you might receive a warning message to the effect that the current version of Navlisp does not match the version of initnavlisp that created your workspace. This is usually not significant. The only compatibility problem that may arise is when you attempt to use a system function in Navlisp that the initnavlisp program did not know about. Occasionally, the changes to Navlisp will create irreconcilable differences, and initnavlisp will have to be re-run. This will, of course, destroy the contents of your current workspace. Therefore, keep your favorite functions in source files. Then they can be restored easily (see getfun for an example of how to do this).

File handling capabilities are embedded in the UNIX operating system. Navlisp gets its input from the standard input file (stdin) and writes on the standard output file (stdout). This can be changed on the invocation line by:

```
% navlisp <notherin >notherout
```

as for any other UNIX process.

For news on the latest developments in Navlisp, display the file /etc/navlispnews. E.g.,

```
% page navlispnews
```

2. Terms used in the descriptions

The following meta-terms will be used in the next section to help describe the objects and functions of Navlisp. Note that these meta-terms are not part of the programming language proper and are not recognized by Navlisp as they are used below (unless the

user defines them so).

=> A symbol used to denote the resulting value or type of a function.

alist Refers to a special data structure used rather commonly in the interpreter:

((obj1 . val1) (obj2 . val2) (obj3 . val3) ...).

It is an 'association list' in that it associates an object with a value. The elements of an assocation list are dotted pairs.

boolean Basically, this refers to one of two values: nil (false) or T (true). However, many functions in LISP recognize any non-nil value as a true value. See member and remprop for examples.

dotted pair The basic element of LISP is the cell, each of which consists of two fields: the car and the cdr fields. Most LISP cells contain atoms or lists in the car field and only lists in the cdr field. Occasionally (see alist, above) it is convenient to store atoms in the cdr field: this cell is then said to contain a "dotted pair". Note that the following two S-expressions are identical, but the first shows the contents of the cells of the list explicitly, while the second is the more usual representation:

(a . (b . (c . (d . nil))))

(a b c d)

exp Used wherever a numeric expression could be used. Implies that the type of the expression must be an integer or real.

list Implies that only lists are acceptable and that atoms are excluded.

a	an atom, not a list
(a)	acceptable, a list
(a (b c))	another good list

number An atom type that subsumes both integers and reals.

object Refers to any entity that can be defined in Navlisp: atoms, numbers, strings, symbolic atoms, or lists (not a mutually exclusive list of entities).

oblist Navlisp keeps a list of all defined symbols. This list is of the form:

(symbol1 symbol2 symbol3 ...)

where each symboli is a plist.

plist — Each element of the oblist is the property list (i.e., plist) of a symbol. This plist is of the form

(sym1 object1 sym2 object2 sym3 object3 ...)

The symi are the attributes, or properties, of the symbol. Unless the user has modified the plist, one of these symi is the property PNAME. Other symbol properties the system recognizes are SUBR, FSUBR, LSUBR, EXPR, FEXPR, LEXPR, FILE, CADRS, VALUE, and TRACE (all of which are defined below). The user may define additional properties through the use of putprop.

S-expr — Denotes a list; a balanced set of parentheses:

a	is an S-expr
(a b)	is an S-expr
(a b))	is not
(a (b c) d)	is an S-expr
(a ((b c) d)	is not

string — Refers to a concatenation of zero or more characters. The functions print and fprint print strings surrounded with double-quotes and with all non-printable characters expanded using the up-arrow escape convention (see the section on input conventions). The functions printf and fprintf do not print the surrounding quotes, nor do they translate the non-printable characters.

symbol — There are basically three kinds of atoms in Navlisp: numbers, strings, and symbolic atoms. A symbolic atom is simply an object on the oblist, which is the 'symbol table' for Navlisp. Note that 'oblist' is a slightly misleading use of the word as not all objects are on the oblist (e.g. numbers and strings).

3. Input conventions

Navlisp's prompt is a curly brace '{'. The interpreter reads an S-expression from the standard input, and evaluates it, and writes the result on the standard output. For convenience, the user may indicate the end of an S-expression with a closing curly brace '}' which will be expanded automatically to close all open sub-S-expressions. Navlisp also implements the square-bracket convention from Interlisp: a right square bracket ']' will produce enough right parentheses to match the most recent left square bracket '['.

```
{ (defun f (p1) (cond [(null p1) (foo (bar nil]
                      (T (foo (car p1]
```

is equivalent to

```
{ (defun f (p1) (cond ((null p1) (foo (bar nil)) )
                      (T (foo (car p1)) )
                      )
                )
```

To imbed arbitrary characters in strings and names follow the UNIX convention: the back-slant character inhibits interpretation of the immediately following character. There are two exceptions: '\n' is the new-line character, and '\t' is the tab character. the null character '\0' may not be imbedded in Navlisp strings or names. For example, ab\ cd would imbed a blank in the name "ab cd". Non-printable characters can also be imbedded in names, so use with caution.

Use the above convention also to put non-printable or control characters in strings (a more likely necessity). In addition, to give the user full access to the control characters, the up-arrow escape convention is used. For example, '\t' == '^I', and '\n' == '^J'. (The up-arrow convention is used when printing control characters in strings printed with print or printf.) The up-arrow is escaped with the backslant.

```
{ (printf "ab\ncd" CR)
ab
cd


{ (print "ab\ncd" CR)
"ab^Jcd"


{ (printf "The length of ^J is "
      (stringlength "^J") CR) ; non-printable
The length of
 is 1


{ (print "The length of \^J is "
      (stringlength "\^J") CR)
The length of ^J is 2
```

4. Error messages

Error messages in Navlisp are of the form

```
object cName: * Message *
```

where Name is a mnemonic indicating where the error was discovered. Most often Name will be the function that detects the error. Occassionally it will be the name of an internal (to the interpreter) function. The message will indicate what the problem was and hopefully give enough information that the problem can be corrected. The character prefix to Name differentiates the various messages within a function. If an object can be printed that would be helpful it is displayed before the message proper.

Messages that begin "Sys err:" are just that and should (ideally) never occur. However, if one does occur, please note the circumstances and report them.

5. Alphabetic listing of Navlisp objects and functions

Each of the following entries is in the format:

name type (syntax) => returned type or value

where type is usually SUBR, FSUBR, LSUBR, system symbol, or one of a few other miscellaneous types. Square brackets indicate optional items and should not be confused with the square bracket input convention described above. Numbers appended to names are for identification purposes only.

' input convention
The single quote is used as an input convention for the quote function (q.v.).

* LSUBR (* exp exp ...) => number
Returns the products of the expressions. Times is a synonym.

+ LSUBR (+ exp exp ...) => number
Returns the sum of the exp's. Plus is a synonym.

- LSUBR (- exp1 exp2 exp3 ...) => number
Returns exp1 minus exp2 minus exp3 ... . Difference is a synonym.

. system symbol
Used in input and output to indicate that the cdr of a cell is not a list but an atom. Care should be taken to always surround the dot of a dotted pair with blanks as it is legal to imbed dots in names and numbers.

```
{ (cons 'x 'y)
(x . y)

{ (cons 'x '(y))
(x y)

{ (cdr '(x.y))
nil

{ (cdr '(x . y))
y

{ (setq x.y 'something)
something

{ (setq x . y 'something)
' aRead: * ')' expected at end of dotted pair's cdr *

{ x.y
something
```

/ LSUBR (/ exp1 exp2 exp3 ... expn) => number
Returns (...((exp1/exp2)/exp3)...)/expn . Quotient is a synonym.

< LSUBR (< exp1 exp2 exp3 ...) => boolean
Returns T if exp1 < exp2 < exp3 ... . Lessp is a synonym.

= SUBR (= object1 object2) => boolean
If object1 and object2 have the same values, then equal returns T, otherwise nil. Equal is a synonym.

> LSUBR (> exp1 exp2 exp3 ...) => boolean
Returns T if exp1 > exp2 > exp3 ... . Greaterp is a synonym.

0,1,2,... SUBR (integer list) => object
Returns the nth element of the list. That is

```
(1  '(a b c)) == (car list)   => a
(2  '(a b c)) == (cadr list)  => b
(3  '(a b c)) == (caddr list) => c
(4  '(a b c)) => nil, as does 5, 6, etc.
(0  '(a b c)) => (a b c)
(-1 '(a b c)) => (a b c) so do negative numbers
```

It is not an error to apply the nth function to nil, as it will only return nil. However, it is still an error to take the car or cdr of an atom other than nil. (TRACE INTEGERS) will trace all of the integer functions: integer functions cannot be traced individually (e.g. (TRACE 1) will not work). Also note the

following:

```
{ (setq n 1)
1

{ (n '(a b c))
n dInterpreter: * Unknown function in eval *

{ ((eval n) '(a b c))
a
```

abs SUBR (abs exp) => number
Returns the absolute value of the numeric expression exp.

add1 SUBR (add1 exp) => number
Adds one to the value of the numeric expression exp.

alphalessp SUBR (alphalessp string1 string2) => boolean
Returns T if, in a character by character comparison using the ASCII collating sequence, string1 is strictly less than string2. Shorter strings are less than longer strings, all else being equal.

and FSUBR (and object1 object2 ...) => object
The objects are evaluated from left to right until one evaluates to nil. Evaluation is discontinued at that point and nil is returned. If none of the objects evaluate to nil, then the value of the last object evaluated is returned as the value of and.

append LSUBR (append list list ...) => list
Creates a new list that has all the elements of the argument lists as members. This is distinguished from the nconc function in that a new list is created and the argument lists are not changed.

```
{ (append '(a b c) '(d e))
(a b c d e)

{ (setq x '(a b))
(a b)

{ (eq (append x '(d e)) x)
nil

{ (eq (nconc x '(d e)) x)
T
```

apply LSUBR (apply fcn args [alist]) => object
The function fcn is applied to the arguments args in the context/environment alist, if specified.

```
{ (apply 'cons '(a b))
(a . b)
```

ascii — SUBR (ascii integer) => string
Returns a string consisting of the single character corresponding to the integer parameter. E.g. (ascii 10) returns a string consisting of a carriage-return.

assoc — SUBR (assoc object alist) => dotted pair
Returns the dotted pair whose car is the object and whose cdr is the value associated with that object. If no such object is on the alist, assoc returns nil. Uses equal for the equality comparison on the association list.

assq — SUBR (assq object alist) => dotted pair
Returns the dotted pair whose car is the object and whose cdr is the value associated with that object. If no such object is on the alist, assq returns nil. Uses eq for the equality comparison on the association list.

atom — SUBR (atom object) => boolean
Returns T if object is an atom. There are several kinds of atoms to be distinguished:

integer — 16-bit, two's complement

reals — 32-bits; corresponds to C's float

string — a variable length concatenation of characters

symbol — anything on the oblist, including nil

boundp — SUBR (boundp symbol) => boolean
Returns T if symbol is in the current environment. If symbol is not on the current alist, then if it has a VALUE attribute/property, boundp returns T. Otherwise, boundp returns nil. Note that if the VALUE of the symbol is nil, that is considered unbound.

```
{ (setq x nil)
nil

{ (boundp 'x)
nil

{ (let ((x nil)) (boundp 'x))
T
```

break — system symbol
Used in cond (q.v.) to implement Parnas' it-ti construct.

bye — SUBR (bye) => nothing
The workspace is saved and the process is exited, returning the user to the process that invoked Navlisp.

CADRS — system symbol
A property name that indicates this symbol is an extended c*r; e.g. cadr, caddr, cdadadadddaar, etc. There can be up to 16 d's and a's in between the c and the r. The value of the CADRS attribute is a binary encoding of the d and a sequence.

car — SUBR (car (object1 . object2)) => object1
Equivalent to first. Returns the first element of the list. If the argument is not a list, car reports an error. (car nil) is currently an error, although there is some debate as to whether it should simply return nil (see the integer functions 0, 1, 2, etc.).

cdr — SUBR (cdr (object1 . object2)) => object2
Equivalent to rest. Returns all but the first element of the list. If the parameter is not a list, cdr reports an error. (cdr nil) is currently an error, although there is some debate as to whether it should simply return nil (see the integer functions 0, 1, 2, etc.).

chdir — SUBR (chdir dirname) => boolean
A UNIX-specific command that changes the default directory. Dirname must evaluate to a string. Chdir returns T if the command was successfully executed, and nil otherwise. The directory name must be less than 256 characters.

cond — FSUBR (cond (p1 s1) (p2 s2) ...) => object
Navlisp's conditional expression evaluation function. The pi's are predicate expressions evaluating to a boolean value. The si's are sequences of S-expressions. The cond in Navlisp is a partial implementation of David Parnas' it-ti construct, an extension to the normal LISP cond. The pi are evaluated until one evaluates to a non-nil value. The si associated with this non-nil pi are evaluated in a left to right order. The last element of the si list is one of the keywords break or repeat (if neither is present, break is assumed). The former exits the cond command, the latter repeats it.

```
{ (let ((i 0))
       (cond ((lessp i 5) (printf i " ")
                          (setq i (add1 i))
                          repeat
                          )
             (T (terpri) break)
       )
)
0 1 2 3 4
nil
```

cons SUBR (cons object1 object2) => list (or dotted pair)
Constructs a 'dotted pair' whose car is object1 and whose cdr is object2. The following identities hold:

```
(eq (car (cons x y)) x)            => T
(eq (cdr (cons x y)) y)            => T
(cons 'a 'b)                       => (a . b)
(cons 'a (cons 'b (cons 'c nil))) => (a b c)
(cons 'a '(b c d e f))             => (a b c d e f)
```

CR system symbol
This symbol has a VALUE property whose value is a new-line character. Note that the VALUE for this symbol is not a string consisting of the single character, new-line, but is rather another symbol whose PNAME is a new-line character. This allows CR to be used by all the print functions and still serve as a new-line character. Otherwise, if the VALUE were a single character string, the results would not be pleasing:

```
{ (print "string1" CR "string2" CR)
"string1" "^J" "string2" "^J" "^J"
```

when what we really want is

```
{ (print "string1" CR "string2" CR)
"string1"
 "string2"
```

DEBUGGC SUBR (DEBUGGC) => boolean
Primarily for the developer's use, this causes a garbage collection each time a cell is allocated from the free list. If you really want to slow your program down ...

defun FSUBR (defun [type] fcnname ([parlist]) body) => fcnname
FSUBR (defun fcnname [type] ([parlist]) body) => fcnname
Defines a lambda expression with the name fcnname; type must be one of EXPR, FEXPR, or LEXPR, with EXPR the default. If type is FEXPR or LEXPR then the parameter

list should have only one formal parameter. Any previously existing FEXPR, LEXPR, or EXPR properties on fcnname's plist are removed before the new one is added.

delete LSUBR (delete object list [integer]) => list
A very bizarre function from Maclisp: it returns a list with all objects equal to object deleted. (If the integer is present, then only the first n objects are deleted.) As a side effect, it also removes all objects equal to object from the original list unless the object is the first in the list, in which case the first occurence of object is not removed from the original list. Some examples will help clear this up:

```
{ (setq x '(a b a c a b a d a c a b a d))
(a b a c a b a d a c a b a d)

{ (delete 'a x 5)
(b c b d c a b a d)

{ x
(a b c b d c a b a d)    ; note the first a
```

Note that delete uses equal for its equality test.

delq LSUBR (delq object list [integer]) => list
Same as delete but uses eq for its equality test.

difference LSUBR (difference exp1 exp2 exp3 ...) => number
Returns (exp1 - exp2 - exp3 - ...). '-' is a synonym. It is equivalent to (- (+ exp1 exp2 ...)).

divert LSUBR (divert which name [string])

which — Must evaluate to one of the atoms input, output, or append. If append is used, the output is diverted and appended to the end of the named file.

name — Evaluates to a symbolic atom whose value will be UNIX's file descriptor.

string — An optional expression that evaluates to a string that is the UNIX recognized file name. If no string is specified, the PNAME of name is used as the file name.

```
{ (divert 'input 'inf "in.file")
```

This example diverts the input from the current input file to ./in.file (the UNIX name), but which will be

referenced in Navlisp via the atom inf.

```
{ (divert 'input 'in.file)
```

This example is the same as the previous, except that the atom in.file is the atom by which the file is referenced within Navlisp, and whose PNAME will also be the UNIX-recognized file name.

else — system symbol
A 'pretty' symbol that is optional in a cond. It helps to make the code more visible and is ignored:

```
(cond ((eq x y) then (fcn ..) (fcn ...) )
 else ((eq x z) then (fcn ..) (fcn ...) )
 else   :
 )
```

EOF — system symbol
A special object that is returned by I/O routines when end-of-file is reached.

eq — SUBR (eq object1 object2) => boolean
If object1 and object2 are identically the same (not the same value, but the same object) then eq returns T, otherwise nil. Note the following:

```
(eq 'x 'x)                          => T
(eq '(x y) '(x y))                  => nil
(eq (cons 'x 'y) (cons 'x 'y)) => nil
(setq x '(a b))
(setq y x)
(eq x x)                            => T
(eq x y)                            => T
(setq y '(a b))
(eq x y)                            => nil
```

equal — SUBR (equal object1 object2) => boolean
If object1 and object2 have the same values, then equal returns T, otherwise nil. '=' is a synonym.

```
(equal 'x 'x)                                => T
(equal '(x y) '(x y))                        => T
(equal (cons 'x 'y) (cons 'x 'y))            => T
(setq x '(a b))
(setq y x)
(equal x x)                                  => T
(equal x y)                                  => T
(setq y '(a b))
(equal x y)                                  => T
(equal 11 11.0)                              => T
(equal (+ 5 6) 11)                           => T
(equal "str" (implode '("s" "t" "r")))=> T
```

error LSUBR (error object atom ... atom) => nothing
The mechanism by which the user can generate error messages. The first object is printed with a Navlisp error message. This allows the user to print an error message with, perhaps, the offending value or expression. Error uses prints to print the object and printf to print the sequence of atoms. E.g.

```
{ (setq badlist '(a b))
(a b)

{ (error badlist "Pretty bad list: " 'badlist CR)
Pretty bad list: badlist
(a b) aUser: * User generated error *
```

eval LSUBR (eval S-expr [alist]) => object
This is the universal function: the function that can evaluate all other functions, even itself. The S-expr is evaluated in the context of the environment specified by alist. If alist is absent, the S-expr is evaluated in the current context/environment.

```
{ (setq x 1  y 2  z 3)
3

{ (setq alist '((x . 5) (y . 7) (z . 8)))
((x . 5) (y . 7) (z . 8))

{ (eval '(plus x y z))
6

{ (eval '(plus x y z) alist)
20
```

exec FSUBR (exec object1 object2 ...) => integer
The objects in the parameter list (which must evaluate to atoms) are converted into strings and concatenated with separating blanks. The resulting string is then passed to the UNIX shell for execution as a separate process. Exec returns the status. E.g.:

```
(exec 'vi 'file)
(exec 'ls '-l '| 'page)
```

execq LSUBR (execq atom1 atom2 ...) => integer
The same as exec except the parameters are not evaluated before being converted into strings. E.g.:

```
(execq vi file)        same as for exec
(execq ls -l | page)
```

explode SUBR (explode atom) => list
Returns a list of single-character strings that are the

individual characters in the string returned by (string atom). Inverse of implode: (implode (explode atom)) yields (string atom).

EXPR system symbol
A symbol property specifying that this symbol has a user defined S-expression to be evaluated when the symbol appears as the first element of a list. The arguments to an EXPR function are evaluated before they are passed to the function for processing. E.g.

```
(defun f (fx fy fz) ( ... body ...))
(setq x 1   y 2   z 3)
(f x y z)
```

will pass to f the individual arguments 1, 2, and 3.

F system symbol
An equivalent symbol for nil; false.

fclose LSUBR (fclose [name ...]) => T
Closes all or selected files. If no names are given then all files but the standard input and standard output files are closed.

FEXPR system symbol
A symbol property specifying that this symbol has a user defined S-expression to be evaluated when the symbol appears as the first element of a list. The arguments to an FEXPR function are not evaluated before they are passed to the function for processing. An FEXPR function does not receive the individual arguments, but rather a list of the arguments. An FEXPR function can, therefore, have a variable number of arguments. Assume f is an FEXPR function. Then

```
(f x y z)
```

will be invoked with a single argument

```
(x y z)
```

FEXPRs are defined by, e.g.

```
(defun FEXPR fcnname (parm) ( ... body ... ))
```

FILE system symbol
A property symbol whose value is the UNIX file descriptor. Symbols with this property may be used as the first argument to fprint, fprintf, freads, etc.

first SUBR (first (object1 . object2)) => object1
Equivalent to car. Returns the first element of the

list. If the parameter is not a list, first reports an error. (first nil) is currently an error, although there is some debate as to whether it should simply return nil.

fopen LSUBR (fopen mode name [string]) => integer
Opens a UNIX file, where:

mode — Evaluates to one of the symbolic atoms input, output, or append.

name — Evaluates to a symbolic atom whose value will be the file descriptor used by the UNIX file system.

string — An optional expression that evaluates to a string that is the UNIX file name. If no string is specified, the PNAME of the second argument is used as the file name. The filename must be less than 256 characters long.

fprint LSUBR (fprint name object object ...) => object
The objects are written onto the file denoted by the symbolic atom name, which will have been the second argument to an earlier fopen. This function prints atoms and non-atoms. Strings are printed with the double quotes, and each object is separated from its neighbors by blanks. The last object printed is returned as the value of fprint. (cf. print, printf, fprintf)

fprintf LSUBR (fprintf name atom atom ...) => atom
The atoms are written onto the file denoted by the symbolic atom name, which will have been the second argument to an earlier fopen. This function prints only atoms. Strings are printed without the double quotes, and each object is printed adjacent to its neighbors. The last atom printed is returned as the value of fprintf. (cf. print, printf, fprint)

freada SUBR (freada name) => atom
Returns as its value a single atom read from the file name, which was the second parameter to an fopen. (cf. reada)

freadc SUBR (freadc name) => string
The value of this expression is a one character string read from the file denoted by name. The function returns the atomic symbol EOF when end of file is reached. (cf. readc)

freads SUBR (freads name) => object
Reads an S-expr from the file denoted by name, the value of which is set in the second parameter to fopen. (cf. reads)

FSUBR system symbol
A property of symbols that are Navlisp primitives. FSUBR functions expect one argument: a list of the unevaluated arguments. Assume that f is an FSUBR primitive. Then

(f x y z)

will invoke f with the single argument

(x y z)

fterpri SUBR (fterpri name) => CR
Prints a new-line character on the specified file Equivalent to (fprintf name CR). name.

funarg system symbol
See the entry for function below for an explanation.

function FSUBR (function fcn) => (funarg fcn alist)
There is a 'problem' in LISP that Navlisp shares: LISP is a dynamically scoped language, as opposed to a lexically scoped language. This means that the free variables in a function definition are not 'bound' until the function is being evaluated. This may or may not have been what the programmer had in mind. An admittedly contrived example:

```
{ (setq pi 3.14159)
{ (defun perimeter (radius)
        (times 2 (times radius pi)))
    ; later
{ (defun hexagon (radius)
    (let ((pi 3))
        (printf "perimeter of hexagon is     ="
            (times 2 (times pi radius)) CR)
        (printf "perimeter of outer circle is="
            (perimeter radius) CR)
     )
  )
{ (perimeter 4)
25.1327

{ (hexagon 4)
perimeter of hexagon is     =24
perimeter of outer circle is=24
```

Let us assume that the programmer's intention for

perimeter was to calculate the circumference of the circle in which a polygon of a certain radius is inscribed. However, it is also necessary to calculate the circumference of the polygon using its own value of "ratio of perimeter to diameter" (for hexagons it is three). If the programmer insists on calling both values 'pi', confusion will result, especially if this is buried in a large program and not so obvious as in this small example. The problem arose when the programmer thought 'lexical scoping' ("this function will use the global variable/constant") in a dynamically scoped language (which will use the latest definition of an object).

The solution implemented in most LISP systems is to provide a function which tags other functions with the environment in which they are to be evaluated: the third element in the funarg list described above. The funarg symbol at the head of the list is LISP's way of flagging such a construct to the interpreter. By this method, the above problem would be solved by:

```
( (defun auxper (radius)
        (times 2 (times radius pi)))
( (defun perimeter (radius)
        ((funarg auxper nil) radius))
```

This is not very satisfying or transparent, and an enhancement is being considered to either develop a lexically scoped LISP, or allow a switch in Navlisp which will toggle between dynamic and lexical scoping.

gc SUBR (gc) => (gccnt chpair intpair realpair listpair bufpair hwm)
Forces a garbage collection and returns as its value a report on space usage:

gccnt the number of garbage collections since startup

chpair, intpair, realpair, listpair are lists of the form (free max) where free is the number of cells of that type that is free, and max is the maximum available; note that this is the number of cells, not the number of bytes. There are four bytes per character/string cell, two per integer cell, four per real cell, and four bytes per list cell.

bufpair file buffers are allocated on an as needed basis; this pair of the form (free alloc) specifies the number of buffers allocated and not in use and the total number of buffers

allocated. (0 1) implies that there is one file open; (1 1) says there are no files open with one buffer allocated.

hwm — the high water mark, or highest address in use; for addresses greater than 32767 you will have to do ten's complement arithmetic to figure it out as it will print as a negative number.

See MEMORY for an example of use.

gctwa — SUBR (gctwa) => integer
Removes all 'truly worthless atoms' from the object list. A 'truly worthless atom' is defined to be one which has no properties (other than PNAME), and is not referenced by any other object in the system. It returns the number of atoms so removed.

get — SUBR (get symbol1 symbol2) => object
Searches the property list (plist) of symbol1 looking for a property whose name is symbol2. If symbol2 is found on the plist, the value associated with that attribute of symbol1 is returned. Both parameters must be atomic symbols (i.e. on the oblist) or nil is returned. If symbol1 does not have a plist, or if symbol2 is not on symbol1's plist, then nil is returned.

```
the form of a plist for a symbol is
  (-1 attr1 val1 attr2 val2 attr3 val3 ... )
therefore
  (get symbol attri) => vali
```

getfun — LSUBR (getfun fname [string]) => object
If the file string (or the print name of fname, if string is not given) is not open it is opened. An S-expr is read and evaluated, and the result is returned. If the expression is a defun the function will then be defined as a side effect. The symbol EOF is returned at end of file (be careful of other S-expressions that may return the symbol EOF when evaluated!). Note that the input file is not closed by getfun. Therefore, the next call on getfun will read the next S-expression from the file. Such a file of personal function definitions can then be read with

```
(defun LEXPR getall (x)
  (let ((temp) (fd (1 x)) (str (2 x)))
       (cond ((eq (setq temp (getfun fd str)) EOF)
                  (fclose fd) (terpri)
                  break)
             (T (cond ((atom temp) (printf temp " "))
                      (T (print CR temp CR)))
                repeat)
             )
       )
  )
```

If the evaluated S-expr reads from the standard input during its evaluation, the input will come from the same file fname, and not the current stdin.

get_pname SUBR (get_pname symbol) => string
Returns the print name of symbol, which must be an object on the oblist. Note that '_' is the underscore character on most terminals, but may be a back-arrow on some.

greaterp LSUBR (greaterp exp1 exp2 exp3 ...) => boolean
Returns T if exp1 > exp2 > exp3 ... . '>' is a synonym.

implode SUBR (implode (atom1 atom2 ...)) => string
Returns a string that is the result of concatenating the values of (string atomi) for each atom i in the argument. It is the inverse of explode, with the additional capability of accepting multi-character strings in the parameter list.

IGNEOF SUBR (IGNEOF) => boolean
Normally, an end-of-file on the standard input will act as if the (bye) command had been entered. Toggling IGNEOF will change this to simply report the occurence of an end-of-file. Note that control-D from the keyboard is equivalent to an end-of-file.

input system symbol
Used in fopen, divert, and undivert to indicate that the file mode is for reading.

INTEGERS system symbol
Used to denote the integer functions to the TRACE command. The individual functions cannot be traced, but (TRACE INTEGERS) will report each use of any integer function.

intp SUBR (intp object) => boolean
Returns T if the object is an integer. Note that if the object is real (e.g. 23.0) it is not considered an

integer even if its fractional part is zero.

it — FSUBR (it (p1 s1) (p2 s2) ...)
An equivalent name for cond.

label — FSUBR (label fcnname S-expr) => object
Associates fcnname with the S-expr on the alist. In this way, the S-expr can reference itself. Usually it is better just to use defun and associate fcnname with the S-expr globally.

lambda — primitive (lambda (p1 p2 ...) S-expr) (arg1 arg2 ...) => object
The object returned is the value obtained by evaluating S-expr with all occurences of pi replaced with the corresponding argi.

last — SUBR (last list) => list
Returns a list which has as its single element the last element of the argument list.

length — SUBR (length list) => integer
Returns the number of elements in the list.

```
{ (length '(a b (c d)))
3
```

lessp — LSUBR (lessp exp1 exp2 exp3 ...) => boolean
Returns T if exp1 < exp2 < exp3 ... . '<' is a synonym.

let — FSUBR (let par-list stmt1 stmt2 ...) => object
where par-list is of the form

```
((symbol1 object1) (symbol2 object2) ...)
```

All objects are evaluated in the current environment. Then the symbols are associated with the respective values and added to the environment. Then the statements are evaluated. This is Navlisp's form of a local block. The let statement returns the result of the last evaluated statement as its value.

```
{ (setq x '(a b c))
(a b c)

{ (let ((x '(d e f))) (print x CR))
(d e f)


{ x
(a b c)
```

LEXPR system symbol
A symbol property specifying that this symbol has a user defined S-expression to be evaluated when the symbol appears as the first element of a list. The arguments to an LEXPR function are evaluated before they are passed to the function for processing. An LEXPR function does not receive the individual arguments, but rather a list of the arguments. An LEXPR function can, therefore, have a variable number of evaluated arguments. Assume f is an LEXPR function. Then

```
(setq x 1  y 2  z 3)
(f x y z)
```

will be invoked with a single argument

```
(1 2 3)
```

LEXPRs are defined by

```
(defun LEXPR fcnname (parm) ( ... body ... ))
```

LINEIN system symbol
whose VALUE is the currently executing command stored as a list.

list LSUBR (list object object ...) => list
Returns a new list containing the objects as its elements:

```
: (setq x '(a b c))
(a b c)

: (list 'x x '(d e f) (plus 3 5))
(x (a b c) (d e f) 8)
```

LSUBR system symbol
A property of symbols that are Navlisp primitives. LSUBR functions expect one argument: a list of the evaluated arguments. If f is an LSUBR primitive. Then

```
(setq x 1  y 2  z 3)
(f x y z)
```

will invoke f with the single argument

```
(1 2 3)
```

mapcar LSUBR (mapcar fcn list1 list2 ... listn) => list
If listi = (eli1 eli2 eli3 ... elim) then mapcar returns

```
                list1 list2   listn

                  v     v       v
(list (fcn el11 el21 .. eln1)
      (fcn el12 el22 .. eln2)
        :
      (fcn el1m el2m .. elnm) )
```

Obviously, fcn must be an n-argument function. If the lengths of the lists are unequal, the shortest list determines the number of elements in the returned list (and therefore the number of evaluations of fcn). The fcn used in mapcar should probably not be an FSUBR or FEXPR as the results may be unexpected.

max LSUBR (max exp exp ...) => number
Returns the value of the expression with the maximum value.

member SUBR (member object list) => list
If the object is in the list then the remainder of the list starting with object is returned. If the object is not a member of list, then nil is returned. Note that the equality comparison is made by equal.

```
(member 'x '(a b m r x z))      => (x z)
(member 'x '(a b (x z) 2))      => nil
(member '(x z) '(a (x z) b 2))  => ((x z) b 2)
```

MEMORY SUBR (MEMORY integer1 integer2 integer3 integer4) => nothing
Increases the memory allocations for the various types of memory spaces. Navlisp memory is divided into character (or string) space, integer space, real number (floating point) space, and list cell space. There are actually more spaces (file buffer space, garbage collection space, and miscellaneous space) but the user does not have control over these. By using the MEMORY command the user can increase (see SCRUNCH for decreasing) the amount of memory available for characters, integers, reals, and lists. That order corresponds to integer1, integer2, integer3, and integer4 in the command list. The integers must be positive and small enough to not overflow memory. MEMORY does not return a value: it does not return! Calling MEMORY resets the interpreter to prompting at the command level. The following example should help make this clear:

```
{ (gc)
(2 (224 500) (93 200) (100 100) (3224 5000)
          (0 0) 27800)
```

shows that two garbage collections have taken place so

far, that there are 224 free character cells (four characters max per cell) out of 500, 93 free integer cells out of 200, all 100 real number cells are free, 3224 list cells are free out of 5000, there are no file buffers allocated, and the high water mark in memory is 27800. Now allocate 7000 more character cells allowing up to 28000 bytes of character storage:

```
{ (MEMORY 7000 0 0 0)

{ (gc)
(4 (7224 7500) (93 200) (100 100) (3224 5000)
          0 0) -9736)

{ (MEMORY 1000 0 0 0)
 aAdjmem: * Insufficient system memory *
```

memq SUBR (memq object list) => list
Same as member above except that the equality comparison is made by eq instead of equal.

```
(memq 'x '(a b m r x z))      => (x z)
(memq 'x '(a b (x z) 2))      => nil
(memq '(x z) '(a (x z) b 2))  => nil
```

min LSUBR (min exp exp exp ...) => number
Returns the minimum value of the expressions.

minus SUBR (minus exp) => -exp
Negates the value of exp.

minusp SUBR (minusp number) => boolean
Returns T if the number is less than zero (negative).

mkatom SUBR (mkatom string) => symbol
If there exists an object on the oblist whose print name is string, then that object is returned. Otherwise, a new object with the name string is added to the oblist. Note that it is possible to define objects this way that are very difficult to access:

```
{ (mkatom "a b")
a b

{ (remob "a b")
"a b"  aRemob: * Not a symbolic atom *

{ (remob (mkatom "a b"))
nil
```

nconc SUBR (nconc list1 list2) => list1
List2 is appended to list1 (via rplacd) and the new (modified) list1 is returned. This is distinquished

from the append function in that nconc changes list1 to produce the new list.

```
{ (append '(a b c) '(d e))
(a b c d e)

{ (setq x '(a b c))
(a b c)

{ (eq (append x '(d e)) x)
nil

{ (eq (nconc x '(d e)) x)
T

{ (nconc x x)
(a b c a b c a b c ... )
```

will produce a circular list which will put the print routine in an infinite loop. However,

```
{ (append x x)
(a b c a b c)
```

does not.

ncons SUBR (ncons object) => list
Equivalent to (cons object nil).

nil system symbol
The empty list, sometimes written (). Used by boolean functions to indicate false. Also used to indicate the end of a list: every list ends with an empty sub-list.

not SUBR (not object) => boolean
Negates the boolean value represented by the object. Equivalent to the null function.

null SUBR (null object) => boolean
Returns T if the parameter is nil, otherwise returns nil.

numberp SUBR (numberp object) => boolean
Returns T if the object is an integer or a real.

OBLIST SUBR (OBLIST) => oblist
The current oblist is returned. The following function will display each object on the oblist:

```
(let ((ptr (OBLIST)))
  (cond ((null ptr) break)
                (T (print (plist (car ptr)) CR)
                   (setq ptr (cdr ptr))
                   repeat)
                )
  )
```

oddp — SUBR (oddp number) => boolean
Returns T if the number (or its integral part) is odd; otherwise returns nil.

or — FSUBR (or object1 object2 ...) => object
The objects are evaluated from left to right until one evaluates to something other than T, at which point evaluation ceases and that non-nil value is returned. If all values are nil, then nil is returned.

output — system symbol
Used in fopen, divert, and undivert to indicate that the file mode is for writing.

pair — SUBR (pair list1 list2) => list3
Each element of list1 is matched up with each element of list2; each list must have the same number of elements. Note that the LISP 1.5 implementation (and, therefore, this one) reverses the order of the lists.

```
(pair '(a b c) '(1 2 3))
((c . 3) (b . 2) (a . 1))
```

plist — SUBR (plist symbol) => list
Returns the plist for the atomic symbol symbol. If the argument is not an atomic symbol an error results.

plus — LSUBR (plus exp exp ...) => number
Returns the sum of the exp's. '+' is a synonym.

plusp — SUBR (plusp number) => boolean
Returns T if the number is greater than zero, otherwise returns nil.

PNAME — system symbol
The print name property of symbols.

print — LSUBR (print object object ...) => object
The objects are written onto the standard output separated by blanks (there is no blank after the last object). The last object printed is returned as the value. Note that string atoms are surrounded by quotes when using print. (cf. printf, fprint, fprintf)

printf LSUBR (printf atom atom ...) => atom
The atoms (or expressions which evaluate to atoms) are written on the standard output. The atoms are not separated by blanks. Strings are not surrounded by double quotes. The last atom printed is returned as the value of printf. (cf. print, fprint, fprintf)

putprop SUBR (putprop symbol value prop) => value
The property prop (which should be a symbolic atom) is added to the plist of the symbolic atom symbol and is given the value value, which is returned as the value of putprop.

QUIT SUBR (QUIT) => nothing
Returns the user to the process that invoked Navlisp (presumably the shell) without saving the workspace.

quote FSUBR (quote object) => object
This function simply returns its argument, whatever it may be. This is a convenient way to delay evaluation of a list or object. For convenience, the apostrophe is used as a shorthand for the quote function. That is, 'a is equivalent to (quote a); and '(a b) is equivalent to (quote (a b)); etc.

```
(cons (+ 1 2) 5)
(3 . 5)

(cons '(+ 1 2) 5)
((+ 1 2) . 5)
```

quotient LSUBR (quotient exp1 exp2 exp3 ...) => number
Returns (... ((exp1 / exp2) / exp3) ...). '/' is a synonym.

reada SUBR (reada) => atom
Returns as its value a single atom read from the standard input. The function returns the atomic symbol EOF when end of file is reached. (cf. freada)

readc SUBR (readc) => string
The value of this expression is a one character string read from the standard input file. The function returns the atomic symbol EOF when end of file is reached. (cf. freadc)

reads SUBR (reads) => object
The value of this expression is the S-expression read from the standard input. The atomic symbol EOF is returned at end of file. (cf. freads)

realp SUBR (realp object) => boolean
Returns T if the object is a real number; otherwise

nil.

remainder SUBR (remainder number1 number2) => integer
Returns number1 modulo number2. If the numbers are reals, they are truncated to integers before the operation.

remob SUBR (remob symbol) => nil
Removes all properties (except PNAME) for the symbolic atom symbol from the oblist. Use the (gctwa) function to remove these 'truly worthless atoms'.

remprop SUBR (remprop symbol prop) => list
Removes the property prop from the plist for symbol. The list returned is the rest of the property list, beginning with the removed prop.

repeat system symbol
Used in cond (q.v.) to implement Parnas' it-ti construct.

rest SUBR (rest (object1 . object2)) => object2
Equivalent to cdr. Returns all but the first element of the list. If the parameter is not a list, rest reports an error.

RESTORE LSUBR (RESTORE) => nothing
Throws away the current memory-resident workspace and reads the workspace stored in the current workspace file. A planned enhancement is the ability to specify the file from which to read the new workspace. The function is useful only at the command level, as RESTORE resets the interpreter.

reverse SUBR (reverse list) => list
Reverses the order of the elements in the list. Does not modify the argument.

rplaca SUBR (rplaca object1 object2) => object1
A dangerous function to physically alter memory in Navlisp. It replaces (car object1) with object2.

rplacd SUBR (rplacd object1 object2) => object1
A dangerous function to physically alter memory in Navlisp. It replaces (cdr object1) with object2.

SAVE LSUBR (SAVE) => nothing
Writes out the user's workspace. Currently, always uses the default workspace (either .navlisp in the current directory, or the file specified on the invocation line). A planned enhancement is to allow SAVE to accept another filename into which the workspace is written. The function is useful only at the command

level, as SAVE resets the interpreter.

SCRUNCH LSUBR (SCRUNCH) => nothing
The only way the user has of returning unneeded free space is by using SCRUNCH. This function will reduce ALL free lists and memory allocation to the minimum required to run. The user will then need to use MEMORY to increase the size of those free lists needed. The function is useful only at the command level, as SCRUNCH resets the interpreter.

set SUBR (set symbol object) => object
The atomic symbol symbol is updated on the alist to be associated with object. If the symbol is not on the alist, then the property VALUE on the symbol's plist is added/modified to have the value object.

setq FSUBR (setq symbol object [symbol object ...]) => object
The same as set except that symbol is not evaluated: this sometimes saves typing quote marks at the keyboard. Setq is also set up to accept a list of symbols and objects, each symbol receiving the value object. The last object in the list is the returned value of setq.

stdin system symbol
The value of the FILE attribute of this symbol is the UNIX file descriptor for the standard input file associated with the Navlisp process.

stdout system symbol
The value of the FILE attribute of this symbol is the UNIX file descriptor for the standard output file associated with the Navlisp process.

string SUBR (string atom) => string
Returns the string form of the atom. If the atom is a

string — it is returned unchanged

number — the value is converted into a string of ascii characters

symbol — the print name (PNAME) is returned

stringlength SUBR (stringlength atom) => integer
Returns the length of the string form of the atom. Stringlength measures the length of the string returned by (string atom).

stringp SUBR (stringp object) => boolean
Return T if the object is a string; otherwise returns

nil.

sub1 SUBR (sub1 exp) => number
Subtracts one from the value of the numeric expression exp.

SUBR system symbol
A property of symbols that are Navlisp primitives. SUBR functions are passed the results of evaluating each of the arguments. Assume that f is a SUBR primitive. Then

```
(setq x 1  y 2  z 3)
(f x y z)
```

will invoke the function f with the three arguments 1, 2, and 3.

subst SUBR (subst object1 object2 list) => list
A new object is created from list in which all occurrences of object2 in the list (and its sub-lists) are replaced with object1. Note that the eq function is used for the equality test.

```
{ (subst 'a 'b '(a b (a b c)))
(a a (a a c))
```

symbolp SUBR (symbolp object) => boolean
Returns T if the object is on the oblist (i.e., is a symbol, not a list or number, or string); otherwise returns nil.

T system symbol
The symbol whose interpretation is 'true': nil is interpreted to be 'false'.

terpri SUBR (terpri) => CR
Prints a new-line character on the standard output.

then system symbol
A 'pretty' symbol that is optional in cond. It helps to make the code more visible and is ignored:

```
(cond ((eq x y) then (fcn ..) (fcn ...) )
 else ((eq x z) then (fcn ..) (fcn ...) )
 else   :
 )
```

times LSUBR (times exp exp ...) => number
Returns the product of the exp's.

TRACE FSUBR (TRACE symbol1 [symbol2 ...]) => (symbol1 ...)
Places a TRACE property on the plists for the indicated

symbols. If there is an EXPR, FEXPR, LEXPR, SUBR, LSUBR, or FSUBR defined for the symbol in question, then during evaluation a trace will be printed showing the name of the function and the parameters being passed to it.

undivert LSUBR (undivert [which]) => T
Which must evaluate to one of the atoms input, output, or append. If elided, both the input and output will be reset to the standard input and output. THIS FUNCTION DOES NOT CLOSE THE UNDIVERTED FILES! (see fclose)

```
(divert 'output 'outf)
(undivert 'output)
```

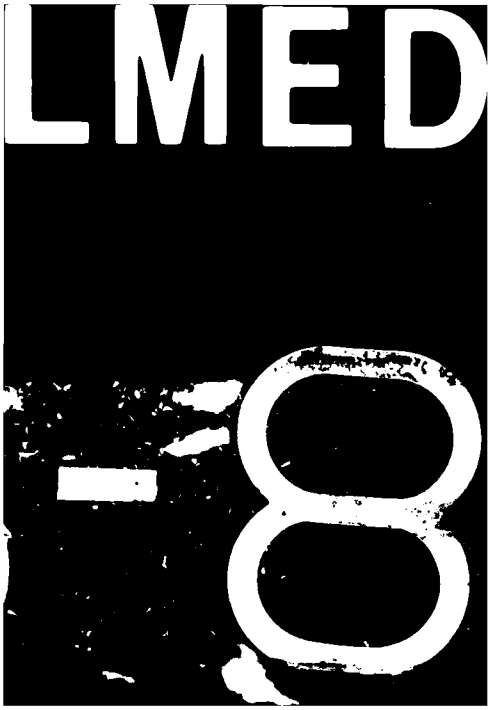
UNTRACE FSUBR (UNTRACE [symbol1 symbol2 ...]) => (symbol1 ...)
Removes the TRACE property from the indicated symbols. If no symbols are specified, then the TRACE property is removed from ALL symbols on the oblist.

VALUE system symbol
The property which holds the value of the symbol.

zerop SUBR (zerop number) => boolean
Returns T if the number's value is zero. Otherwise, returns nil.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93940	2
Office of Research Administration Code 012A Naval Postgraduate School Monterey, CA 93940	1
Chairman, Code 52Bz Department of Computer Science Naval Postgraduate School Monterey, CA 93940	40
A. Dain Samples, Code 52 Department of Computer Science Naval Postgraduate School Monterey, CA 93940	12
Professor Bruce J. MacLennan, Code 52Ml Department of Computer Science Naval Postgraduate School Monterey, CA 93940	1
Professor Douglas R. Smith, Code 52Sc Department of Computer Science Naval Postgraduate School Monterey, CA 93940	1
Professor David Parnas 12503 Davan Drive Silver Spring, MD 20904	1
Chief of Naval Research Arlington, Va 22217	1


LMED
8